Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе № 4**

Дисциплина: Низкоуровневое программирование.

Тема: раздельная компиляция.

Выполнил студент гр. 3530901/10003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.В. Шумилов

(подпись)

Принял старший преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Корнеев

(подпись)

“ ” 2022 г.

Санкт-Петербург

2022

Оглавление.

ТЗ……………………………………………………………………………………...3 стр.

Метод решения………………………………………………………………………3 стр.

Листинг программ………………………………………………………………...…3 стр.

Препроцессирование………………………………………………………………...4 стр.

Компиляция………………………………………………………………....………..6 стр.

Ассемблирование……………………………………………………………………7 стр.

Компоновка…..……………………………………………………………………12 стр.

Создание статической библиотеки и make-файлов…………………..…………16 стр.

Вывод……………………………………………………………………………..…17 стр.

**Формулировка задачи:**

1) На языке C разработать функцию, реализующую определенную вариантом задания функциональность. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке C.

2) Собрать программу «по шагам». Проанализировать выход препроцессора и компилятора. Проанализировать состав и содержимое секций, таблицы символов, таблицы перемещений и отладочную информацию, содержащуюся в объектных файлах исполняемом файле.

3) Выделить разработанную функцию в статическую библиотеку. Разработать make-файлы для сборки библиотеки и использующей ее тестовой программы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

**Вариант задания:** рассчитать заданный член ряда Фибоначчи (вариант 6).

**Метод решения**:

Ряд Фибоначчи:1, 1, 2, 3, 5, 8, 13…

Можем выразить формулой: Fn = Fn-1 + Fn-2, где F1 = 1, F2 = 1

Будем считать члены ряда последовательно, при этом проверяя n. Если n = 1 или n = 2: ответ сразу же 1. В противном случае будем последовательно считать каждый следующий член ряда, проверяя n.

При этом на каждом шаге Fn будет становится Fn-1, аFn-1 - Fn-2.

**Листинг программ:**

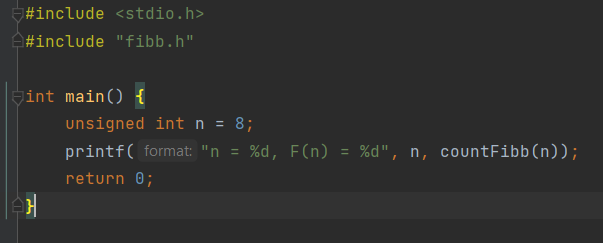
****

Рис.1 файл *main.c*

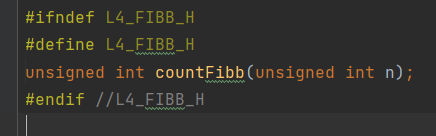


Рис.2 файл *fibb.h*

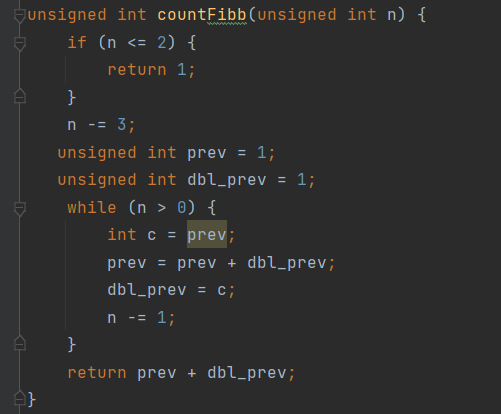
**

Рис.3 файл *fibb.c*

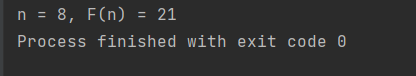


Рис. 4 результат работы программы

В файле “fibb.c” реализована функция countFibb(), в которую передаётся число n – номер необходимого члена ряда Фибоначчи.

Заголовочный файл “fibb.h” содержит в себе определение функции countFibb().

В файле “main.c” мы подключаем заголовочный файл “fibb.h” и используем реализацию функции из “fibb.c”.

**Препроцессирование:**

Начнем сборку программы на C по шагам. Первый этап – препроцессирование файлов с исходным кодом в файлы “main.i” и “fibb.i”.

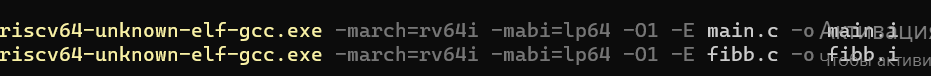
****

Рис. 5 команды для препроцессирования.

Результаты работы процессора практически не отличаются от исходных программ:

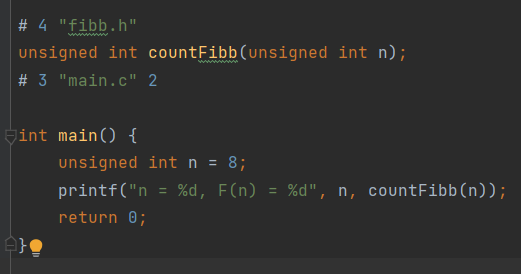


Рис. 6 фрагмент файла *main.i*

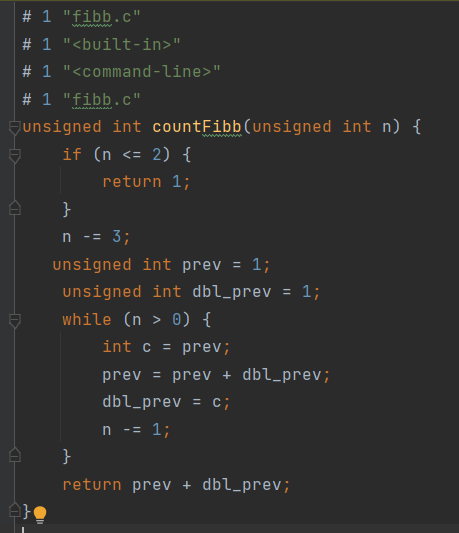


Рис. 7 файл *fibb.i*

Директивы, начинающиеся с символа “#”, используются для передачи информации об исходном тексте из препроцессора в компилятор. Например, строка *# 4 “fibb.h”* в “main.i” означает, что следующая строка является результатом обработки исходного файла “fibb.h”. В данном случае, в основном файле там была строка *#include “fibb.h”.* Т.о., произошла вставка содержимого заголовочного файла “fibb.h”.

В файле “main.i” так же описываются функции, находящиеся в файле-заголовке “stdio.h” и реализованных в стандартной библиотеке C.

**Компиляция:**

Следующим шагом является компиляция файлов *.i* в текст на языке ассемблера “main.s” и “fibb.s”:

****

Рис. 8 команды для компиляции

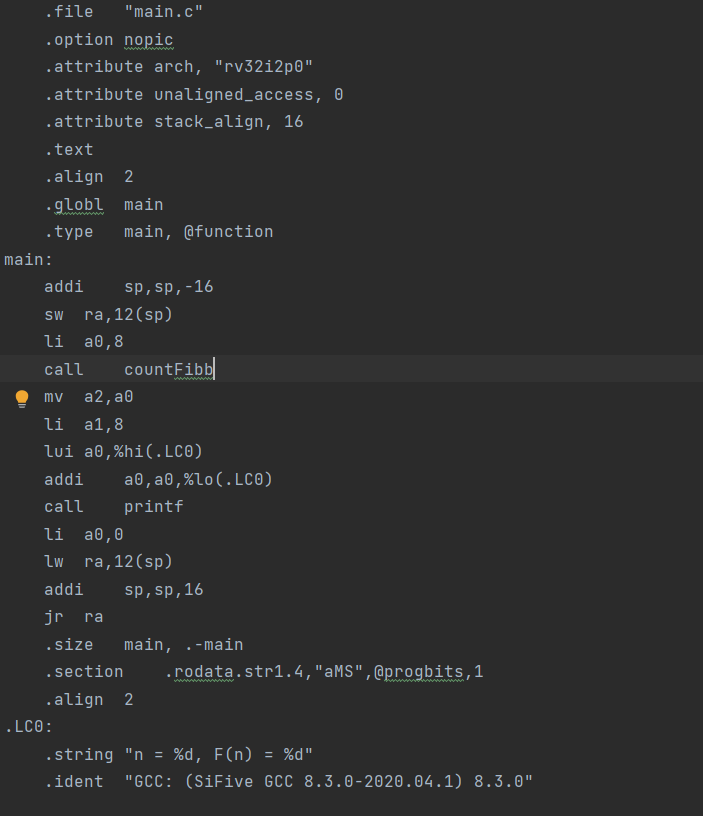


Рис. 9 файл *main.s*

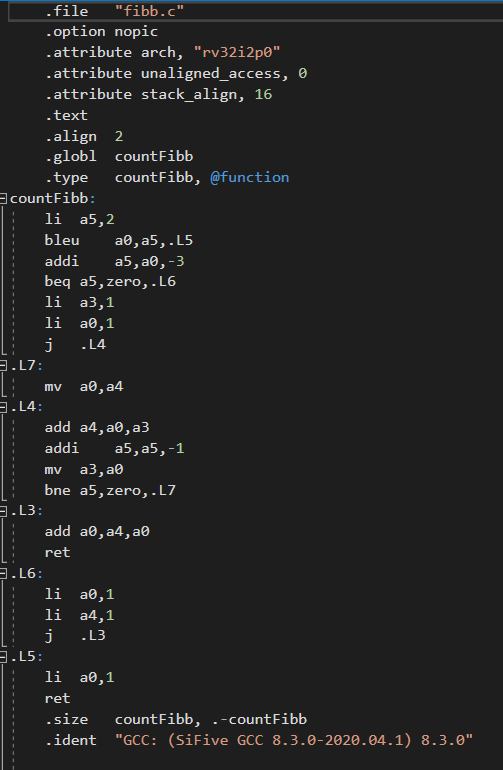


Рис. 10 файл *fibb.s*

Результатом работы компилятора стал код ассемблера RISC-V для соответствующих программ.

В “main.s” в a0 кладется значение n = 8, вызывается функция countFibb.

В “fibb.s” описан процесс нахождения заданного члена ряда Фибоначчи.

**Ассемблирование:**

****

****

Рис.9 команды для ассемблирования

Прочесть полученные в результате ассемблирования файлы просто так не получится. Для этого используем утилиту *objdump*:

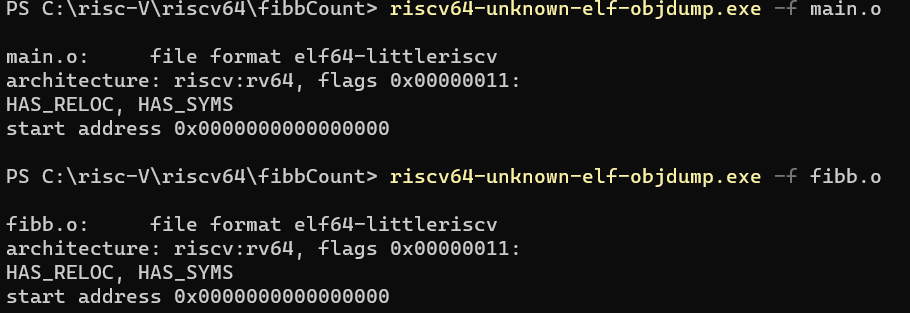


Рис. 10 вывод информации об обоих файлах.

Файлы содержат символы (флаг HAS\_SYMS), содержат таблицу перемещений (флаг HAS\_RELOC). Объектный файл не должен содержать адрес, с которого начинается исполнение программы, такого же формата и исполняемый файл, поэтому здесь это поле равно нулю.

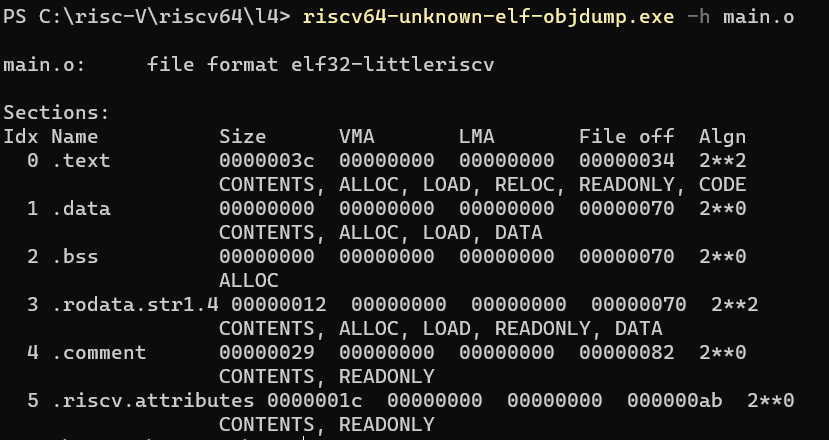


Рис. 10 секция заголовков для файла *main.o*

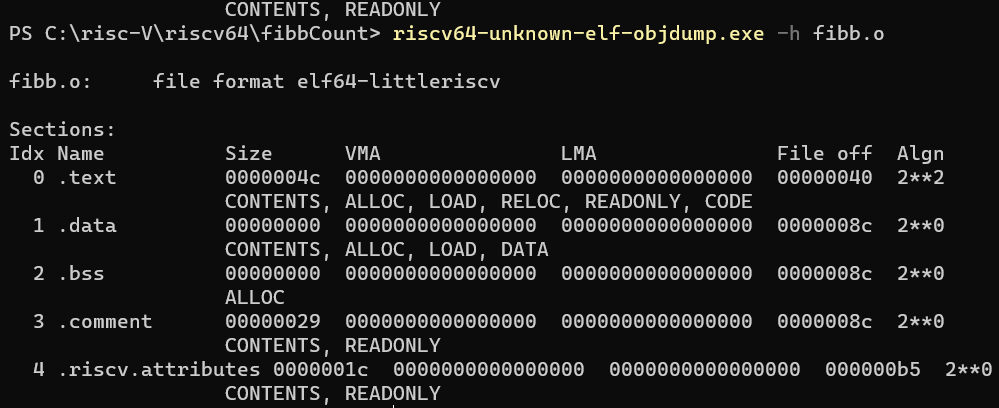


Рис. 11 секция заголовков для файла *fibb.o*

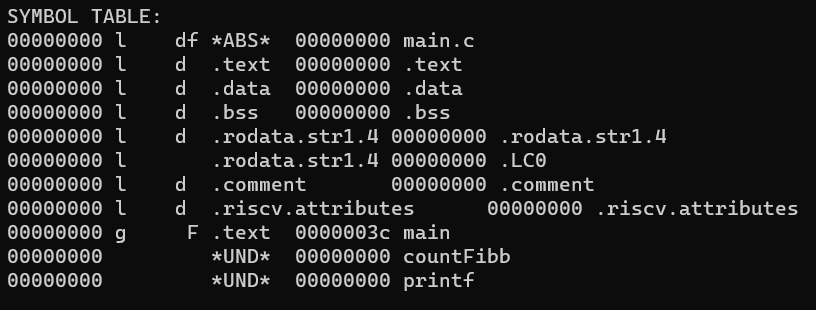
**

Рис. 12 таблица символов *main.o*

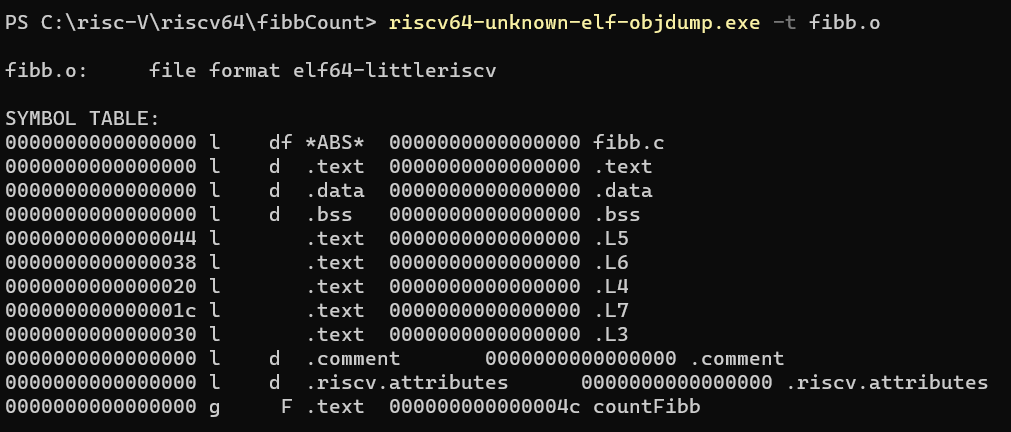


Рис. 13 таблица символов *fibb.o*

Обе таблицы содержат 1 глобальный символ (флаг g) функции (флаг F) – main и countFibb соответственно. Таблица для “*main.o*” содержит также 2 неопределенных символа (UND): countFibb и printf. Эти символы использовались в ассемблерном коде, из которого был получен данный объектный фал, но не были определены. Значит, что символы должны быть определены где-то еще, что и отобразилось в таблице.

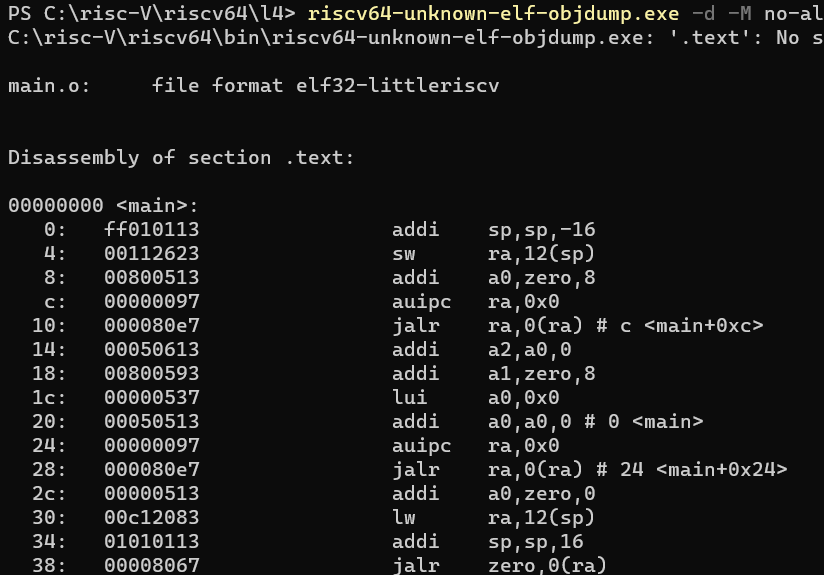


Рис. 14 содержание секции *.text* в файле “main.o” и результат процесса дизассемблирования.

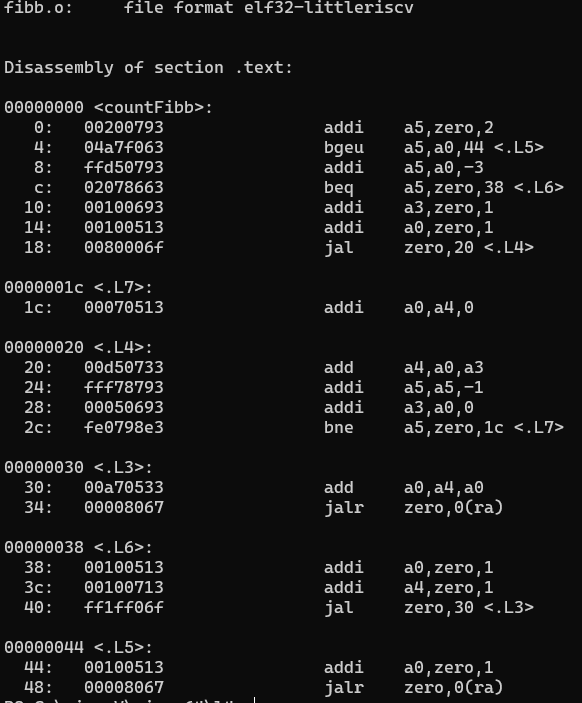


Рис. 15 содержание секции *.text* в файле “fibb.o” и результат процесса дизассемблирования.

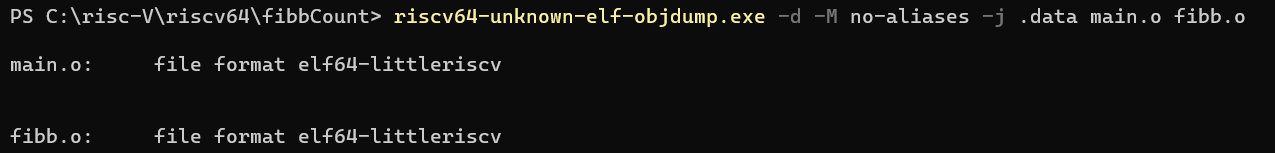


Рис. 16 содержание секций *.data* для обоих файлов.

Секции .data объектных файлов – не содержат данных, размер секций, как было выведено выше (рис. 10 - 11), равен нулю.

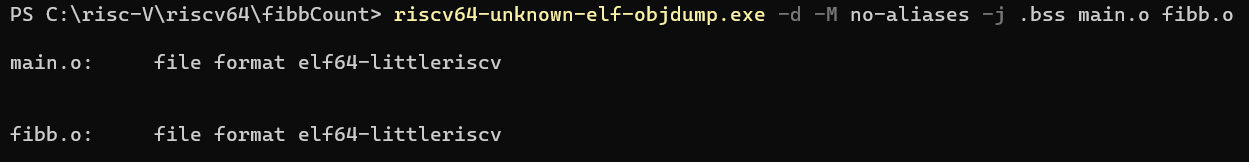


Рис. 17 содержание секций *.bss* для обоих файлов.

Аналогичная ситуация и для .bss секций.

Секция *.comment* – секция данных о версиях – для обоих файлов содержит одни и те же значения – сведения о GCC версии 8.3.0 от SiFive (рис. 18)

Секция *.riscv.attributes* обоих файлов содержит одну и ту же информацию об используемой архитектуре команд RV32I (рис. 19)

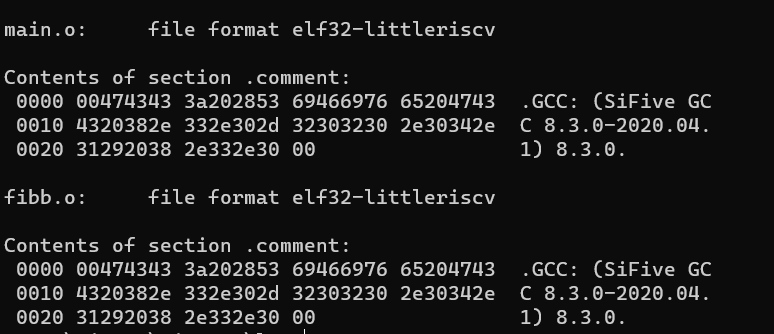


Рис. 18 содержание секции *.comment* для обоих файлов.

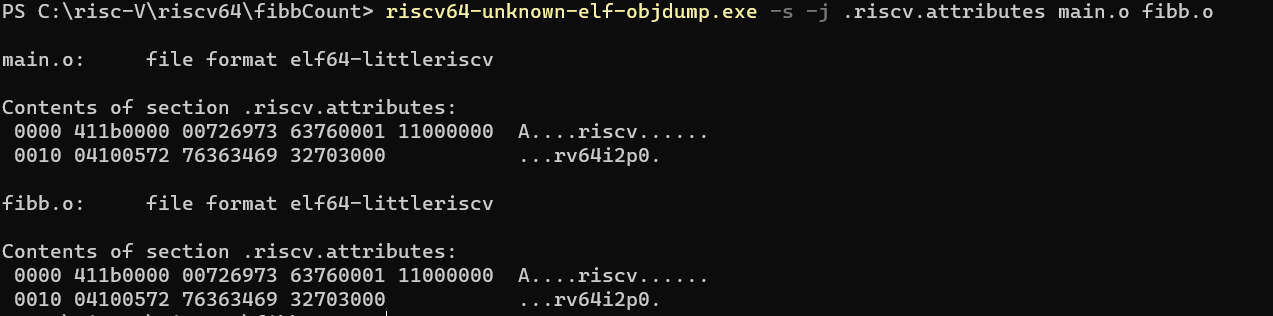


Рис. 19 содержание секций *.riscv.attributes* для обоих файлов

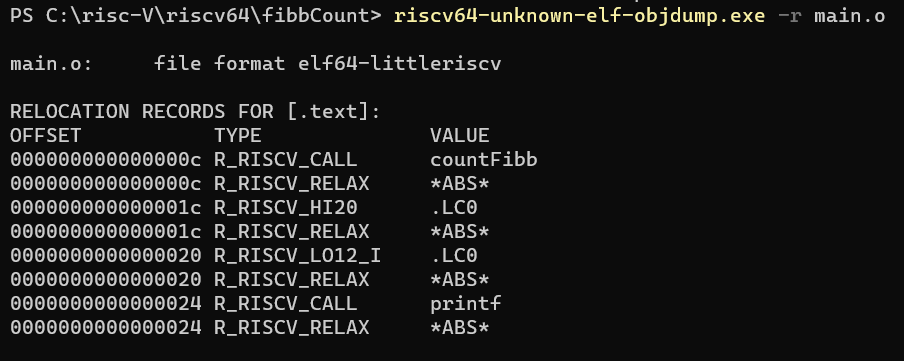


Рис. 20 таблица перемещений для *main.o*

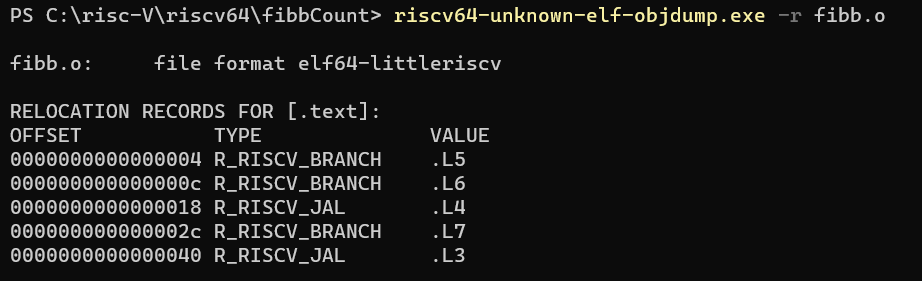


Рис. 21 таблица перемещений для *fibb.o*

**Компоновка:**

Следующим шагом является компоновка и формирование исполняемых фалов программ. Сформированный компоновщиком файл - “a.out”.



Рис.22 команда для компоновки.

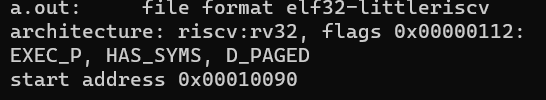


Рис.23 содержимое файла *a.out*

Флаг EXEC\_P указывает, что файл является исполняемым, после загрузки его выполнение должно начаться с адреса 0x00000000000100cc.

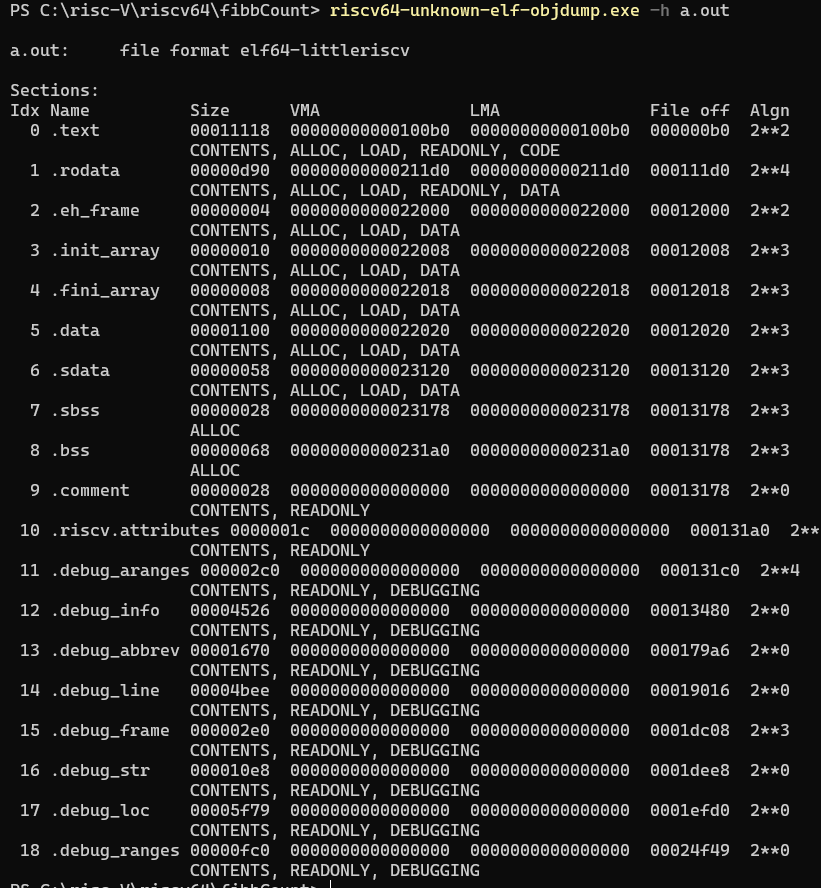


Рис. 24 содержимое файла *a.out*

В исполняемом файле производится слияние содержания секций обоих объектных файлов, а также значительное расширение списка секций новыми блоками.

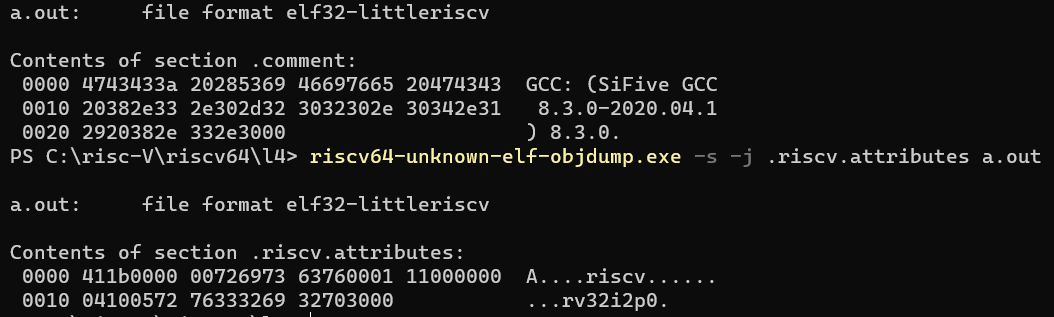
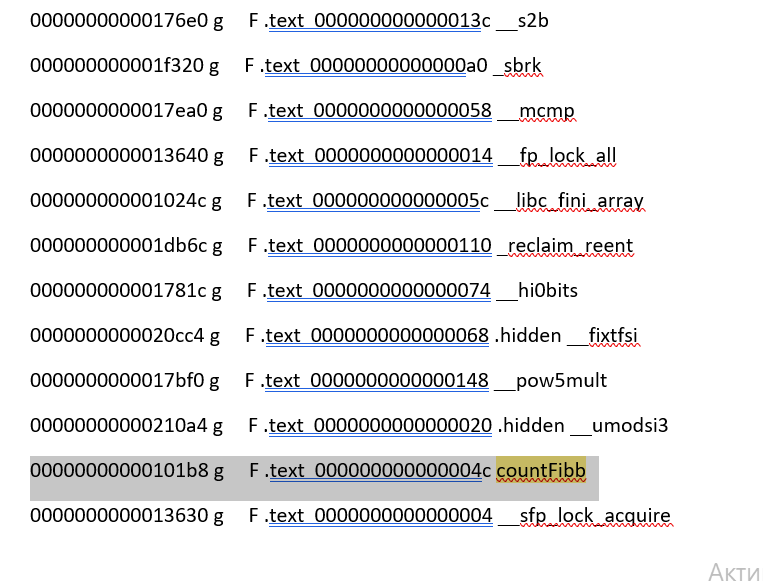


Рис. 25 содержание секций *.comment* и *.riscv.attributes* в файле *a.out*

.comment хранит все те же сведения о GCCверсии 8.3.0 от SiFive

.riscv.attributes по-прежнему содержит информацию об используемой архитектуре команд RV32I



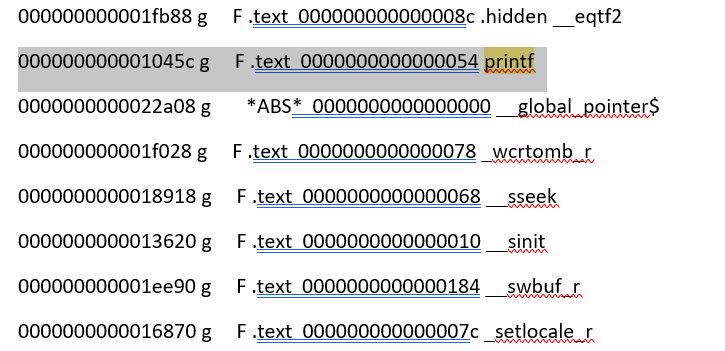


Рис. 26 таблица символов *a.out*

Таблица символов содержит множество дополнительных вхождений, однако определены все нужные секции, метки и адреса. Функции countFibb и printf так же помечены флагом F, но в отличие от стадии ассемблирования, все они являются определенными и содержатся по корректным адресам для успешного вызова этих функций из других участков программ.

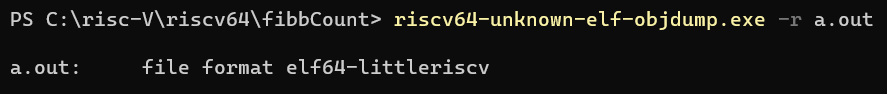


Рис. 27 таблица перемещений *a.out*

Таблица перемещений оказывается пуста, все необходимое было проведено компоновщиком.

Итогом сборки программ на языке C по шагам - исполняемый на процессорах архитектуры RISC-V файл.

**Формирование статической библиотеки, разработка make-файлов для сборки библиотеки:**

Статическая библиотека - объектных файлов, среди которых компоновщик выбирает необходимые для данной программы. Разработанная функция поиска заданного члена ряда Фибоначчи содержится в единственном исходном файле на языке C. Выделим этот файл в статическую библиотеку.

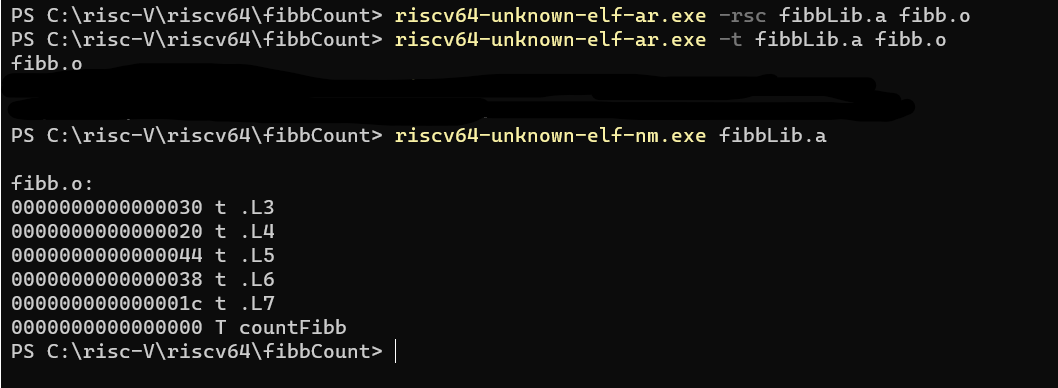


Рис. 28 сборка библиотеки

Код “T” обозначаются символы, определенные в соответствующем объектном файле, кодом “U” - внешние символы. В данном случае, внешних символов нет. Символ функции countFibb - основный символ, определяемым в этом объектном файле, остальные символы определяют лишь локальные для этого файла метки.

Используя получившуюся библиотеку, соберем исполняемый файл.



  
  
  
  
Рис. 29 Сборка и вывод таблицы символов исполняемого файла

Процесс выполнения команд выше можно заменить make-файлами, которые произведут создание библиотеки и сборку программы.

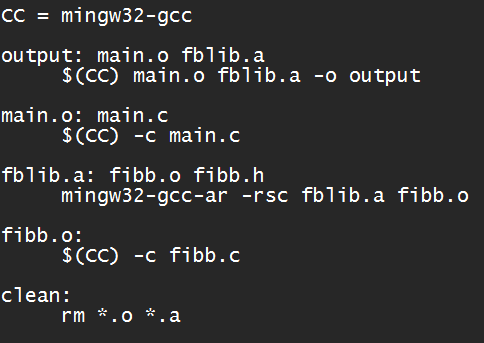


Рис. 30 текст файла “Makefile”

**Вывод:** в ходе выполнения работы была изучена пошаговая компиляция программы на языке С. Также была создана статическая библиотека и произведена сборка программы с помощью Makefile.